# Crawling and analysing python-questions on Stack-Overflow

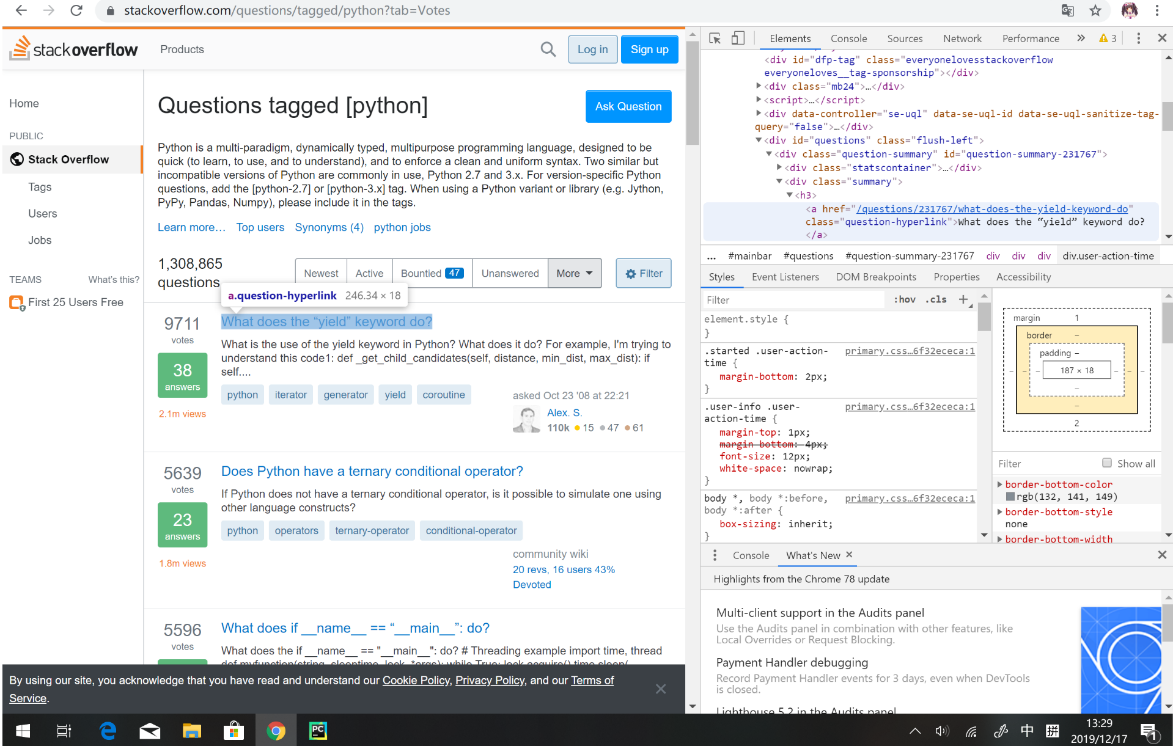
We select this topic to analyse and pave the way for future python learning.

1. **Collecting data from Stack-Overflow by Scrapy**

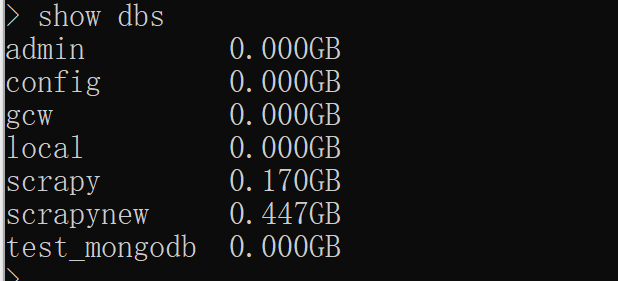
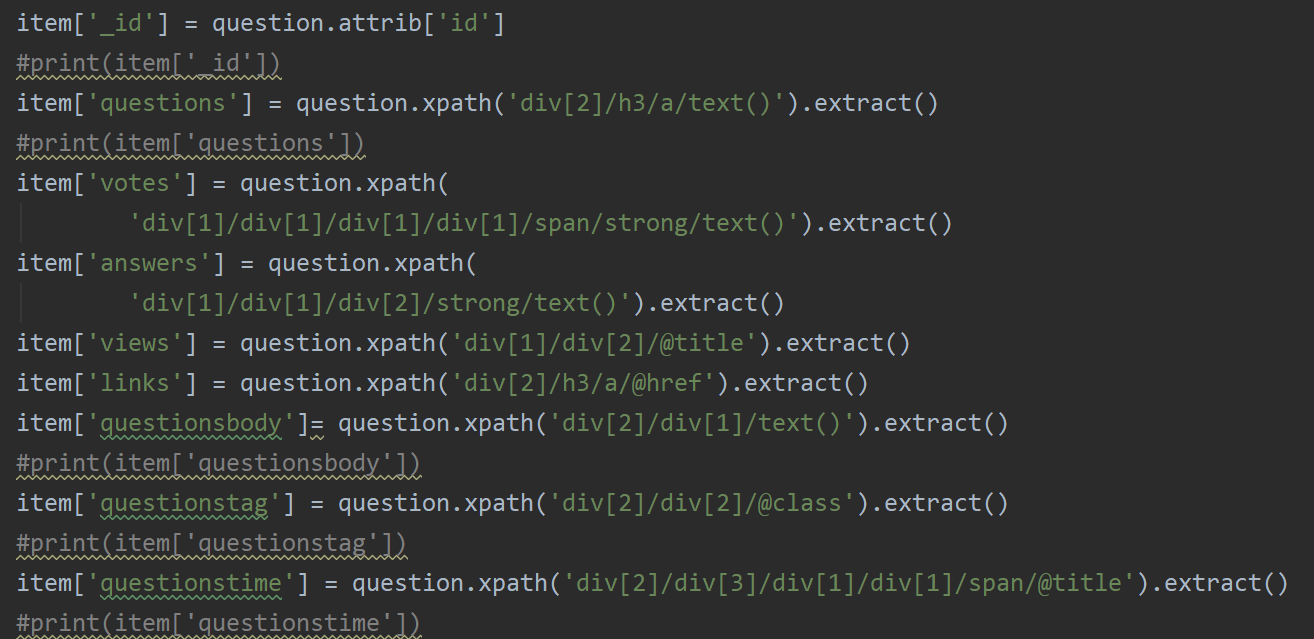
Scrapy is a really commonly used framework and we crawl millions of python questions from Stack-Overflow by Scrapy (If we use API there will have limitation). We also use Redis and MongoDB in our collecting part. (WHY Redis? The master stores the tasks in a queue, and then slaves take the tasks to the hands and analyze the input operations, and store them in a centralized database. This can improve project efficiency.) (Why MongoDB? MongoDB is python friendly and user friendly and can easily accessed using pymongo, if you have used Oracle 11g when you uninstall it you will know how terrible it is). In addition, we do use User-Agent to avoid blocked by their website and we set the DOWNLOAD\_DELAY = 0.7 to crawl faster. (Better 3 or more than 1 to avoid bring too much pressure to their server, but I don’t have too much time).( And also mention that most of the time we may put the store code in pipeline file but I find the code fail to call this part so I just move it to the crawler part.)

First, we can see the URL of Stack-Overflow has unvarying patterns and only change with the number of pages so we use format{} and a for loop to represent all the URL.

And to choose the part of what we want to crawl, we use X-path to find the location of any element we want on the webpage. For example as follow: the question title is in the second div exactly “div[2]/h3/a”, so we can use the location ‘div[2]/h3/a/text()’ to crawl the question title.



And the data we crawl consists of 9 columns as: question’s title, question votes, question’s answer number, question’ view number, question link, question’s body, question’s body (specific question), question’s tag and the question’s asked time. And we can see the size of the data is 447MB in our local MongoDB database with millions of rows.

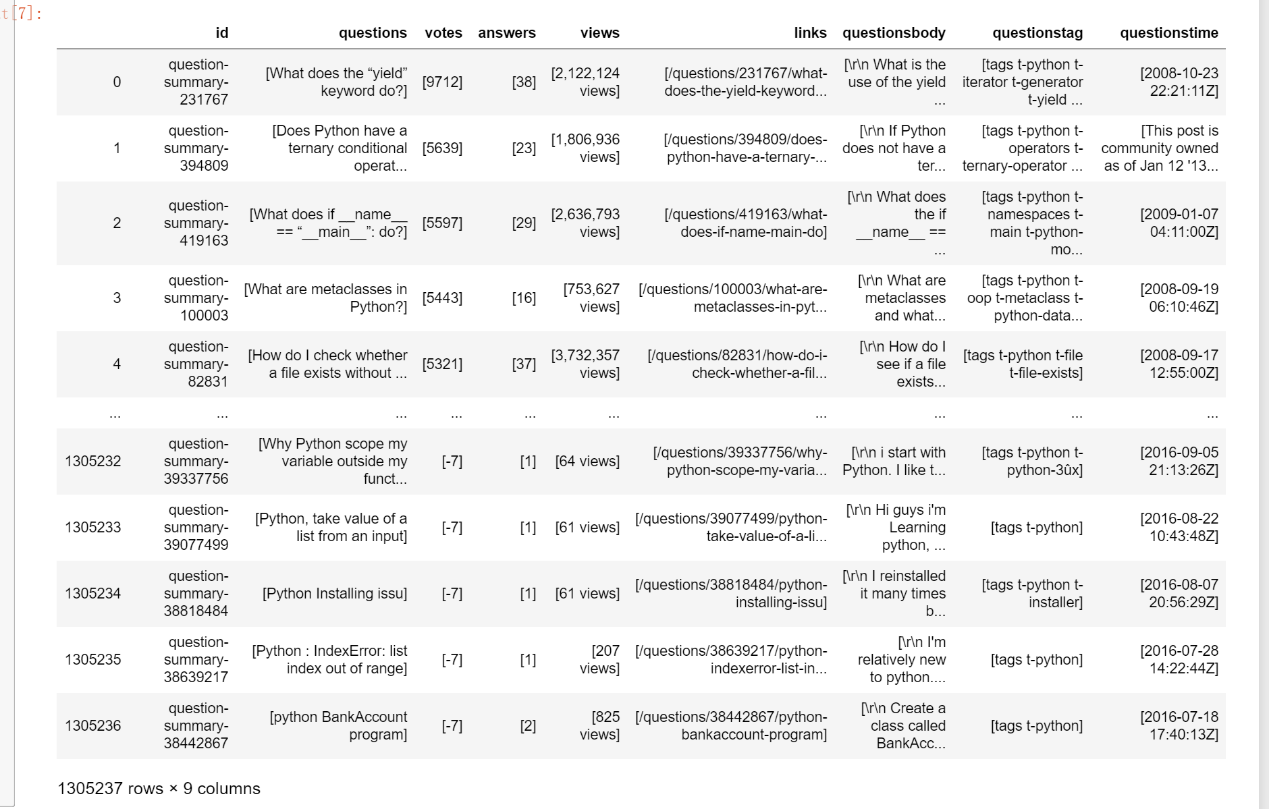


1. **Data cleaning and processing**

Since the data is in my local computer database, we can use pymongo to connect the server and access the data, we read in the data as DataFrame.



The original data:

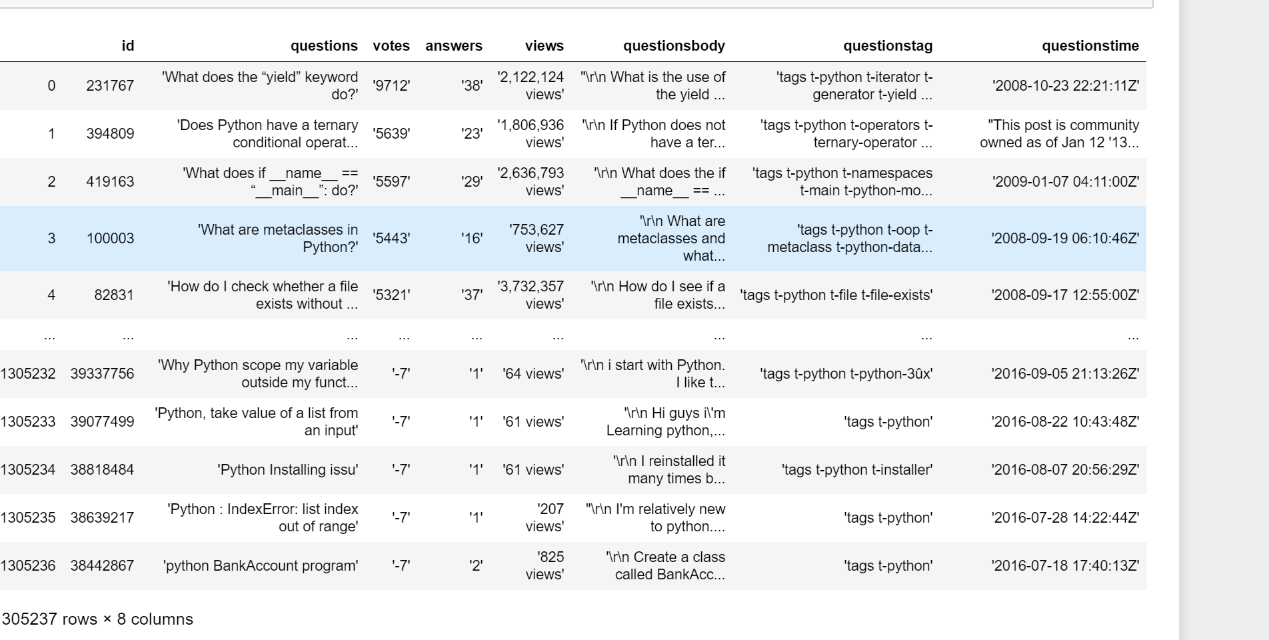


For id: we do slice : df['id']=df['id'].str.slice(17,)

For other columns we remove the unnecessary “[]”.

Drop the [“links”] columns cause we don’t seems need it now.

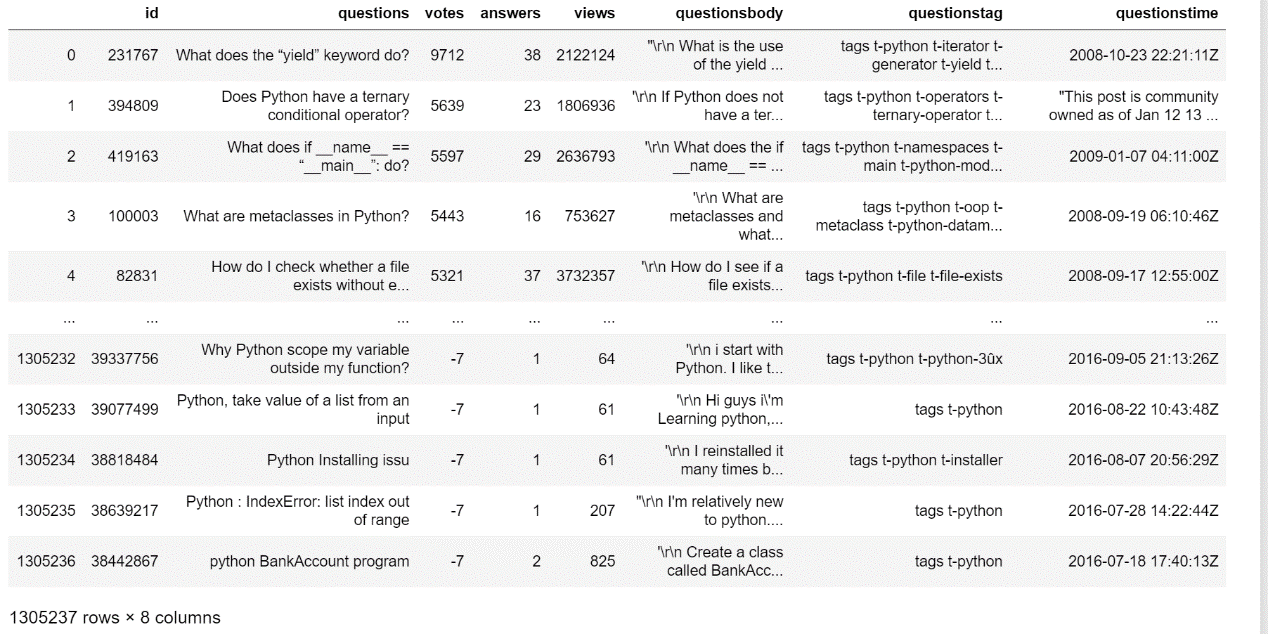
After doing these:



The “ ‘” is still exist, so we still apply lambda function to replace them with “”.

And we also do slice with views to remove the unnecessary string “views” and use lambda function to replace“,” in views number.

Then we change the columns ['id'], ['votes'], ['answers'], ['views'] to numeric number.

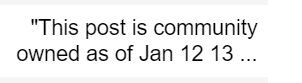


We then rename the ['questionstime'] columns to ['date'], and similarly we slice the only “year/month/day” of ['date'] and use:

**df['date']=pd.to\_datetime(df['date'],format='%Y-%m-%d',errors='coerce')**

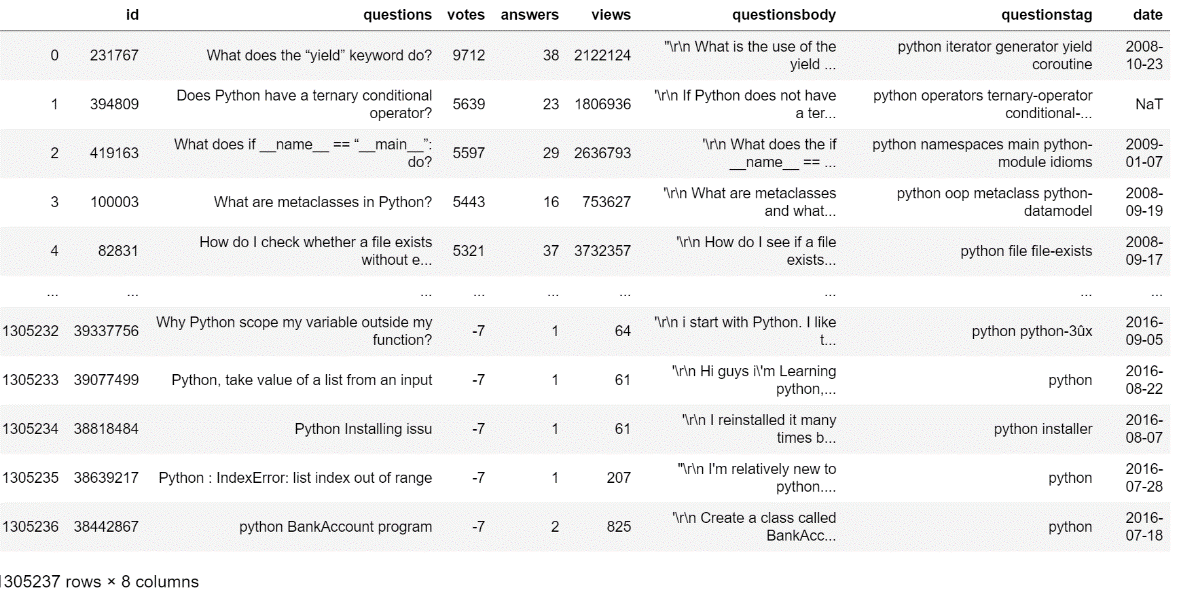
to transfer it to datetime. We fill the errors with NA because there are about 1k questions

datetime is like this:



Maybe some of this questions are not like others asked by users, anyway it’s small number of 1k in millions questions and we will drop the NA datetime question in our time related analysis, but other analysis we still keep it.

And for columns [‘tag’] we also slice the “tag” out and replace the “-t” with “”.

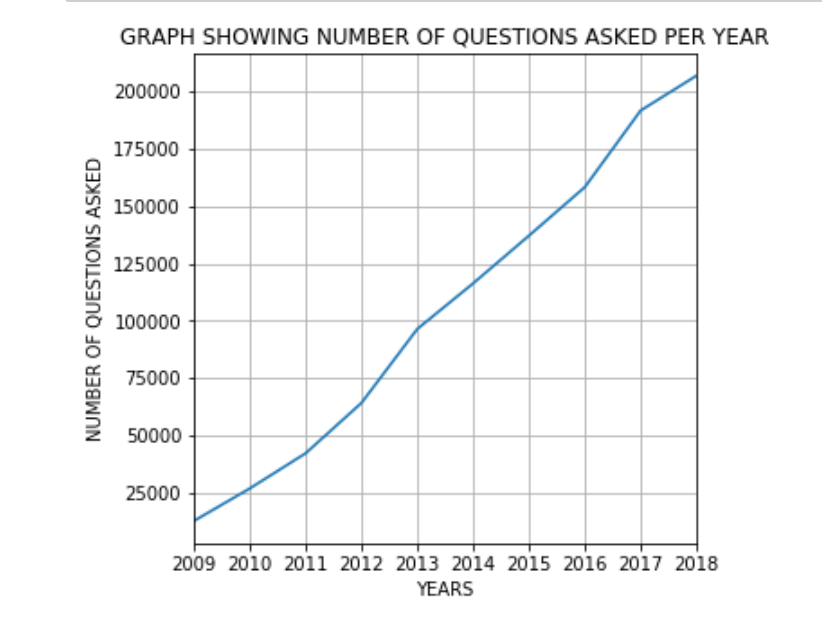


Now it looks much more better.

The columns [‘question’] and [‘questionbody’] and [questiontag] we also do future cleaning in our analysis part.

1. **Data analysis**
   1. **To see the python questions asked by year:**

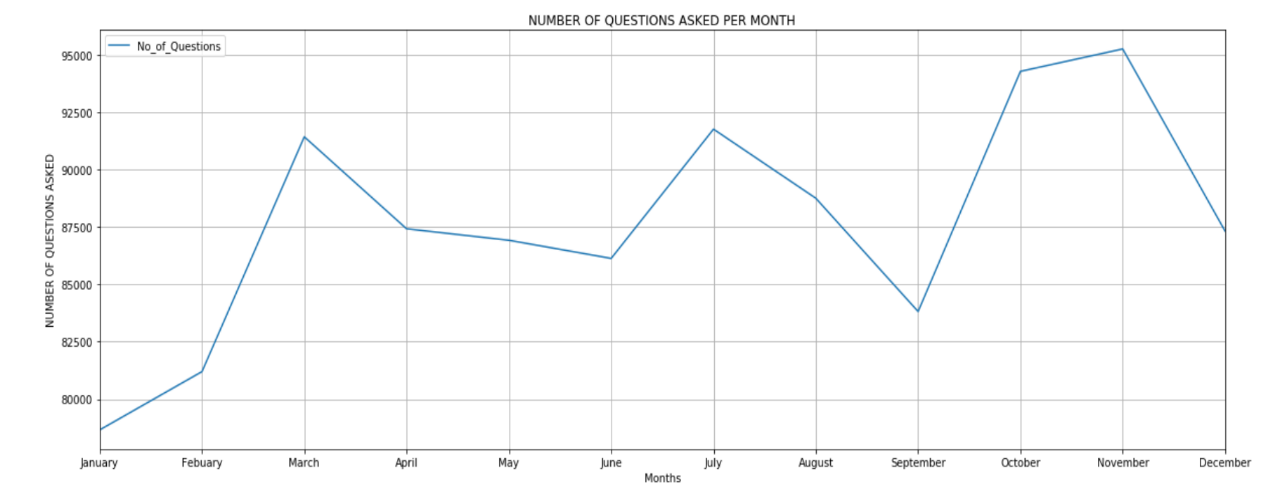
We sort the data by date and can see it start from 2008-08-02 to 2019-12-18 so we consider the 2008 and 2019 data is not complete. We set the date as index and select the only 2009-2018 data and count the values by years then show them by plt as follow:



We can see that the questions of python raised by years and seems that python is more and more popular.

* 1. **To see the python questions asked by year:**

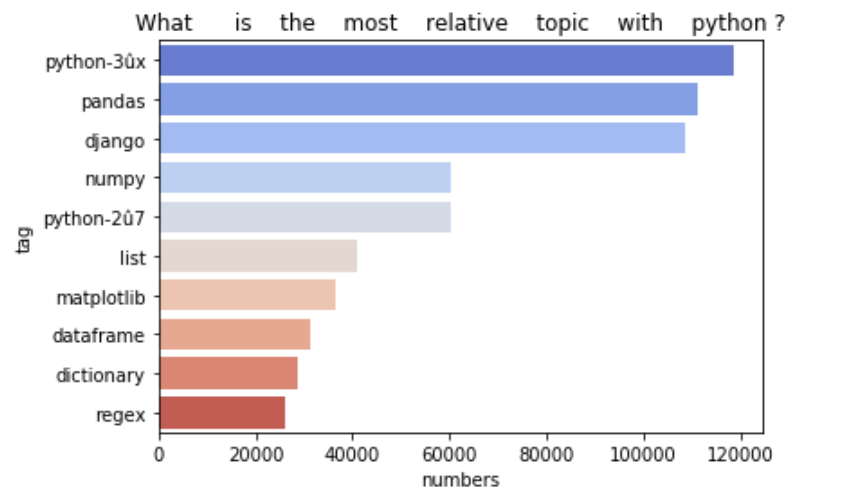
We also use the same data and attribute them to month and find it interesting that in March, July and November there are 3 peaks. We guess there maybe a lot of students users and these 3 month is usually when they do their project or related python homework. And the January is the lowest maybe they go back home to celebrate the New Year Festival…….



* 1. **Tags and python question, what is the most relative topic with python?**

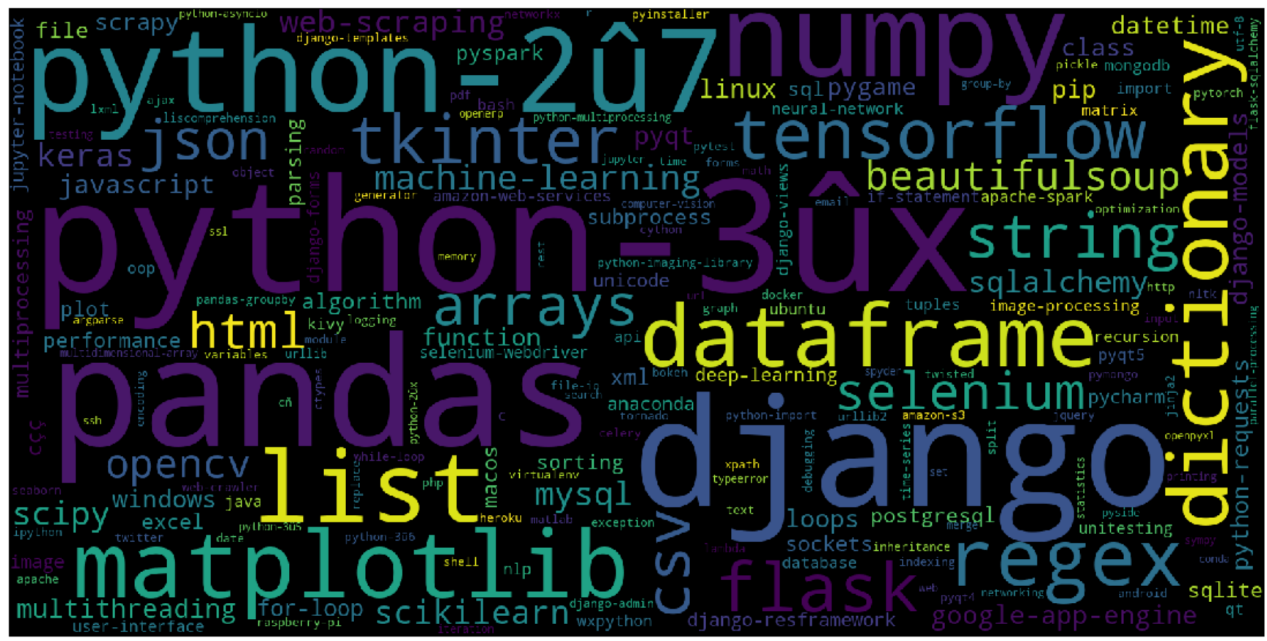
Since the tag of the question is in the same rows, we spilt the tags and the use stack() to combine them. Then we counts the frequency and drop the “python” tag. And use

sns.barplot to show the top 10 related topic as follow:



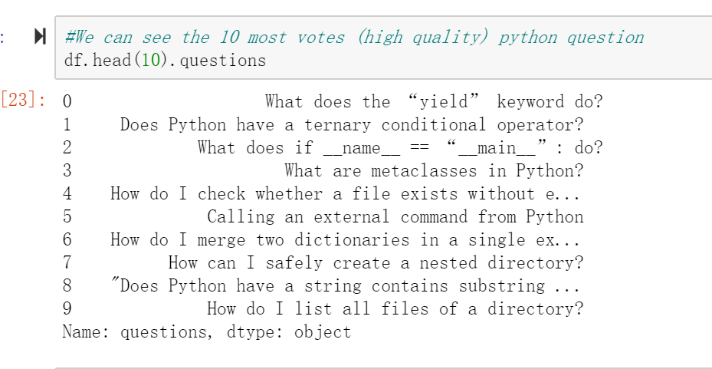
Seems that pandas is really popular question related with python.

We also select the top 200 related topic and show as wordcloud:

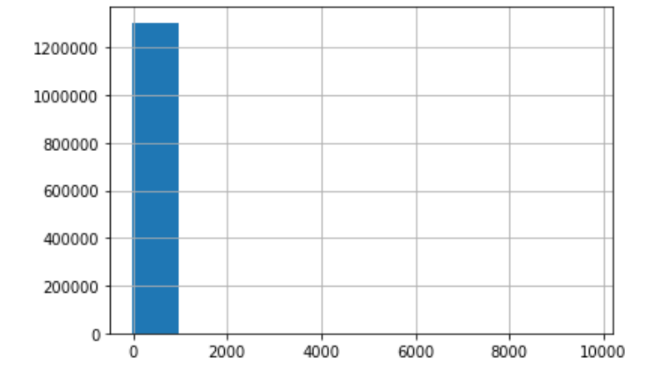


* 1. **Votes and python questions**

We sort the questions by votes and see the top 10 votes questions:

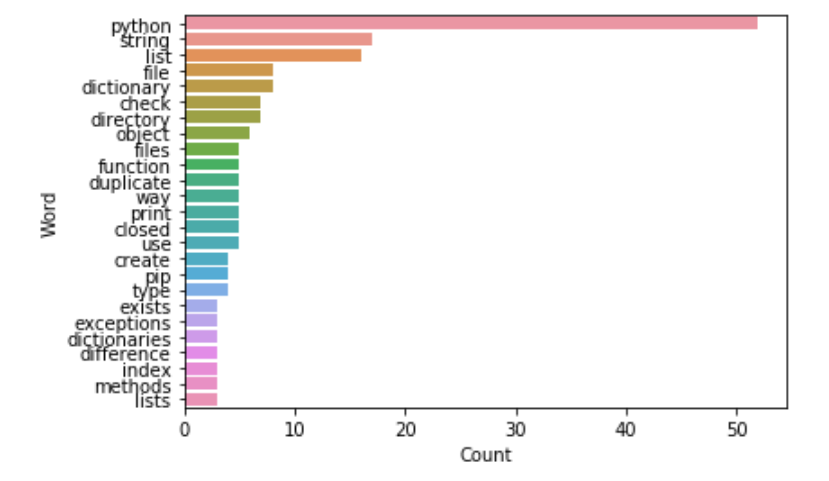
****

To see the distribution of votes, we draw a histogram and see that actually most of the questions is less than 1k votes, so we consider more than 1k as high vote questions.

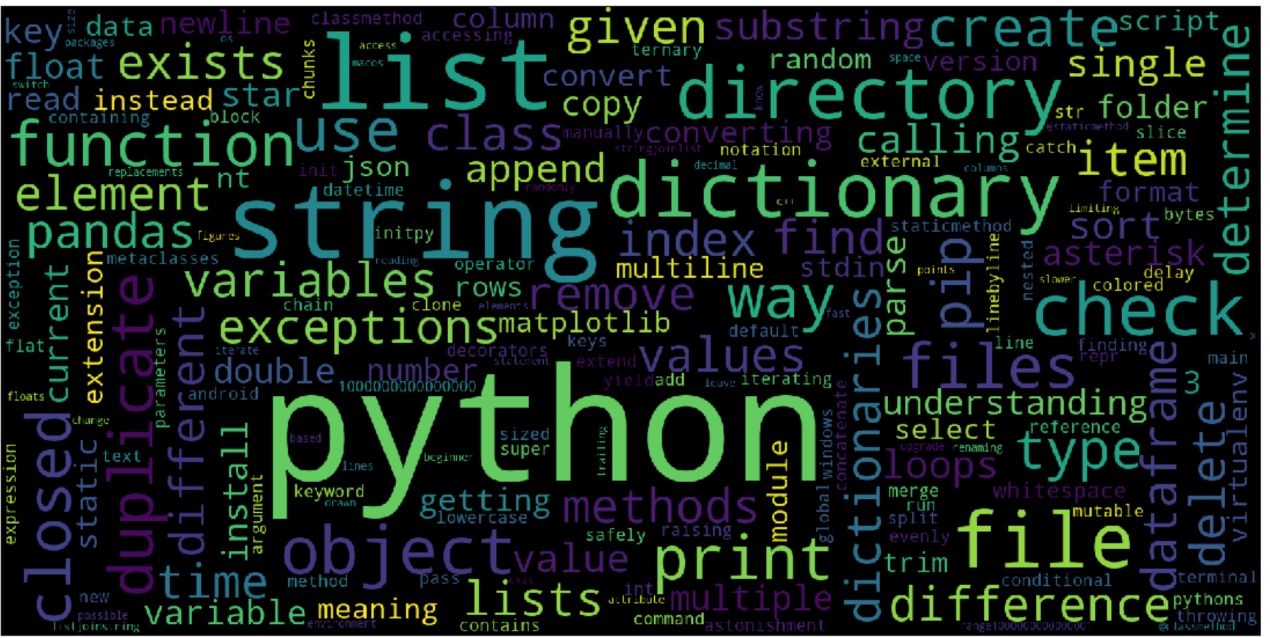
****

So we select the [votes]>1000 questions and see how they ask?

Then we use spacy to clean the question of the title, remove the stop words and punctuation and then same count the only meaningful words. For those meaningful words we use sns.barplot to see the most common 25 words:



And similarly we select the most common words in the most votes questions and generate the word cloud:

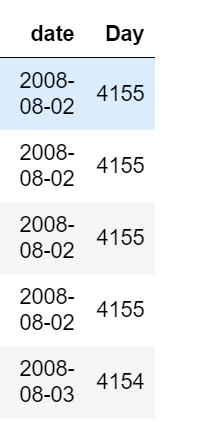


So, how to ask a high quality python question? Maybe: **Using python to store string in a list and store dictionary in file and check?**

* 1. **Do older questions have more votes? And more views? And answers?**

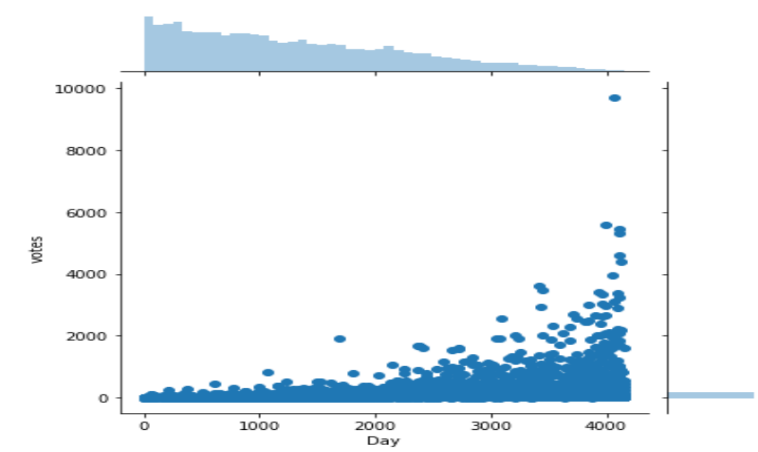
**Votes and views? Answers?**

To find the questions’ existing time and the relation with other we calculate the time by days. We set the start time as the largest in data and minus the date as day then store it as a new column in dataframe.

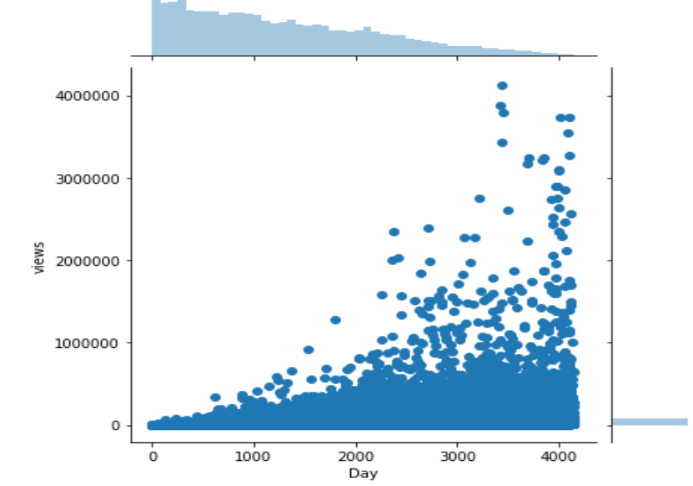


Then we draw the Scatter plot and found that the older questions seem to have more high votes questions.

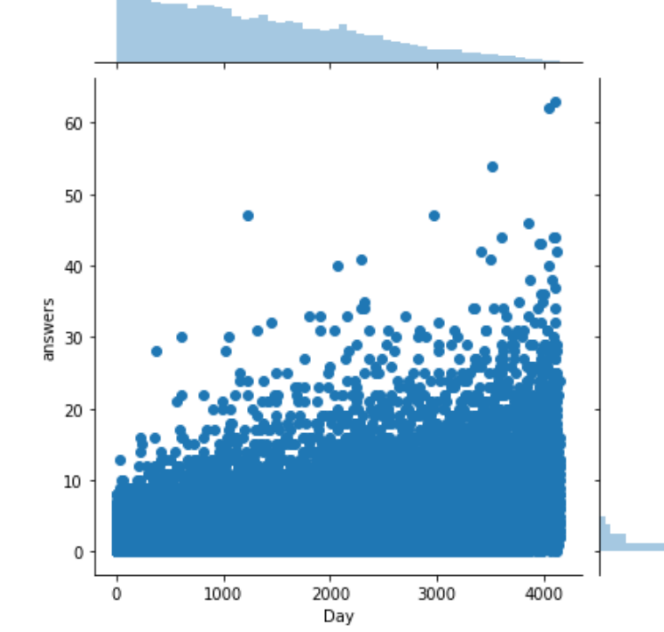
We then calculate the Correlation coefficient and find it is 0.1<x<0.3, so they really have weak correlation.



Same we do the time and views. And also weak correlation.

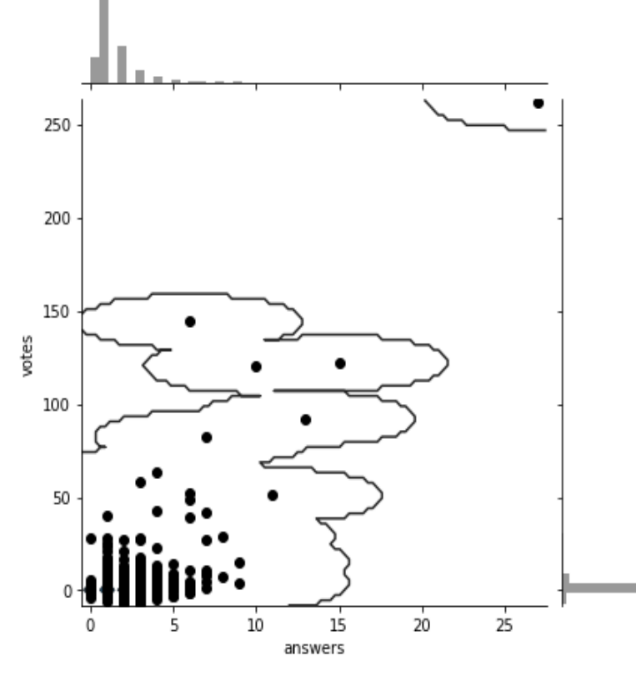
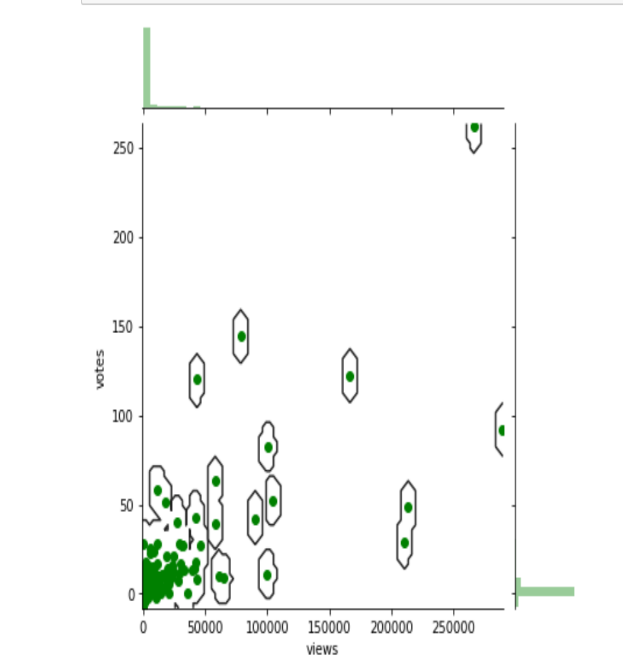
Time and answers: the answers and time do have middle correlation.

And for votes with answers and views:

We do random sampling and do kernel density estimation with scatter plot on sample:

And found that votes have high correction with views and middle correction with answers.

#AI part doing